The Engineering Sketch Pad (ESP): Supporting *Design Through Analysis*

Bob Haimes
haimes@mit.edu
Aerospace Computational Design Lab
Department of Aeronautics & Astronautics
Massachusetts Institute of Technology

John F. Dannenhoffer, III
jfdannen@syr.edu
Aerospace Computational Methods Lab
Mechanical & Aerospace Engineering
Syracuse University

Overview

- Background
- ESP
- Geometry Subsystem (`EGADS` & `OpenCSM`)
  - Architecture
  - Features
  - Distinguishing features
- Analysis Subsystem (`CAPS`)
  - `CAPS` Background
  - Infrastructure
  - Execution
- Closing Remarks

Background

[*Aircraft*] Design is a System Engineer Endeavor

- Given: Requirements & Mission Statement
- Multi-fidelity (traditionally, 3 phases)
- Multidisciplinary/Interdisciplinary
  - Aerodynamics
  - Structural
  - Thermal
  - Controls
  - Costs
  - Manufacturing
  - ...
- Mathematical view: Optimization

Need to be able to realize 3D geometry in order to generate higher fidelity results

Background

Parameterization – Art form

- Describes the *form* and how it can change
- Defines the *Design space* for Optimization
  - Will not be Orthogonal – Will not be Convex
- Should be in a Basis understood by a Practitioner in the Discipline

Design Optimization

- Not just about the final result
  - Optimizers focus on Bad or Incomplete Problem statements by producing *interesting* results
- Learn about the Problem
  - Examine the Optimum
  - Understand the Constraints & the Path taken
- Better designs and better designers!

Background

Off-the-shelf software components

Attempting to build a integrated design system we could use:

- Conceptual Design tools
- Rendering/Artist's Conceptual tools (OpenVSP)
- CAD Systems
  - Catia, SolidWorks, Unigraphics NX, and etc.
- Disciplinary solvers

All components designed and written in **isolation**!

Multidisciplinary Design Optimization frameworks

- OpenMDAO, ModelCenter, Isight (SIMULIA) ...

MDO frameworks as *glue* does not allow for building a design system

Background

Computer-Aided Design (CAD)

- Over the past 40 years, there have been an increasingly-complex (complicated) series of “CAD” systems to support the geometry needs of the manufacturers of mechanical devices – (**mCAD**)
- **mCAD** systems tend to have a single *rendering* of the geometry based on manufacuring *view*
- Need an analysis-*aware* geometry system: **aCAD**
  - Geometry generated at the level of fidelity commensurate with the analysis at-hand and ready for meshing
  - The design has many specific analysis *views*!

Note: “CAD” is sometimes erroneously equated with geometry

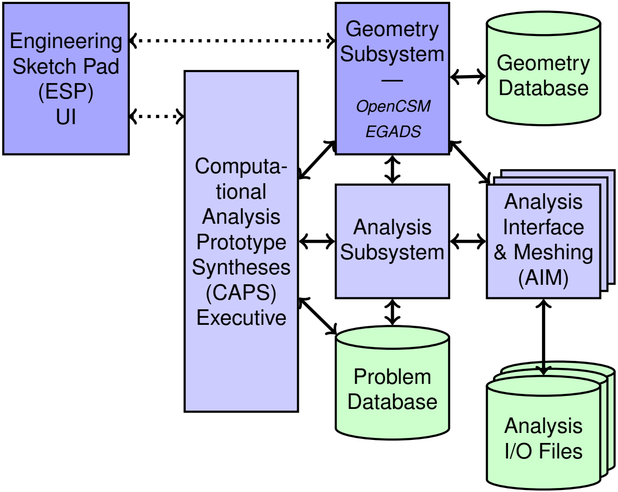
Engineering Sketch Pad (ESP)

ESP is:

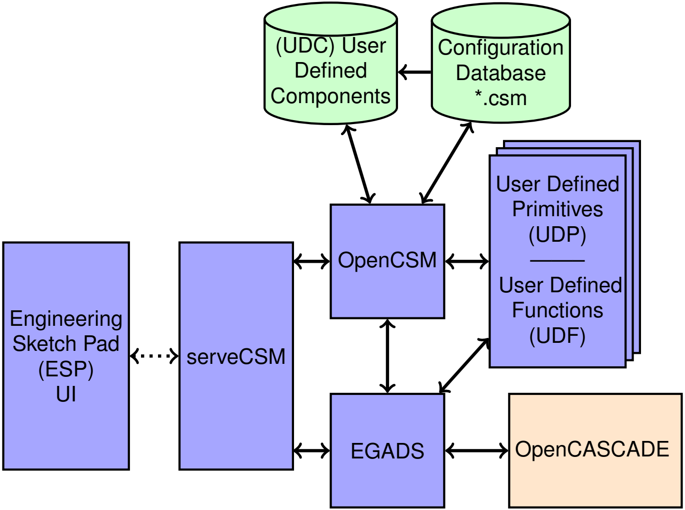
- a geometry creation and manipulation system designed to **fully support** the analysis and design of aerospace vehicles
- a stand-alone system for the development of geometric models
- *layer-cake* of well-crafted open-source APIs
  easily embedded into other software systems to support their geometry and process needs

ESP is not:

- a full-featured mechanical computer-aided design (**mCAD**) system
- a system to be used for creating "drawings"
- an MDO Framework


Engineering Sketch Pad (ESP) UI
Geometry Subsystem
—
OpenCSM
EGADS
Geometry Database
Computa-tional Analysis Prototype Syntheses (CAPS) Executive
Analysis Subsystem
Analysis Interface & Meshing (AIM)
Problem Database
Analysis I/O Files

ESP's Geometry Subsystem Architecture



EGADS

The Engineering Geometry Aircraft Design System (EGADS) is an open-source geometry interface to OpenCASCADE

- reduces OpenCASCADE's 17,000 methods to about 70 C calls
- provides *bottom-up* and/or *top-down* construction
- geometric primitives
  - curve: line, circle, ellipse, parabola, hyperbola, offset, Bezier, BSpline/NURBS
  - surface: plane, spherical, conical, cylindrical, toroidal, revolution, extrusion, offset, Bezier, BSpline/NURBS
- solid creation and Boolean operations (*top-down*)
- provides persistent user-defined attributes on topological entities
- adjustable tessellator (*vs* a surface mesher) with support for finite-differencing (for parametric sensitivities)

The dependency on OpenCASCADE is being reduced while the EGADS API is being maintained

EGADS Topology

Boundary Representation – BRep

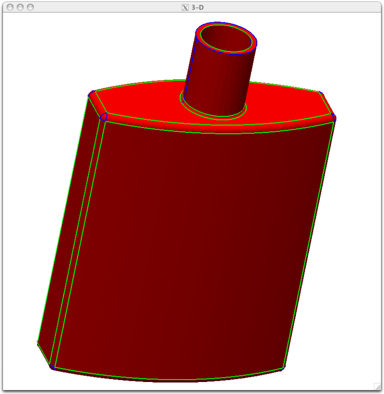
	Topological Entity	Geometric Entity	Function
Top Down ↓	Model		
	Body	Solid, Sheet, Wire	
	Shell		
↑	Face	**surface**	$(x, y, z) = \mathbf{f}(u, v)$
	Loop		
	Edge	**curve**	$(x, y, z) = \mathbf{g}(t)$
Bottom Up	Node	**point**	

ESP works with a *stack* of Body (and/or Node) Objects

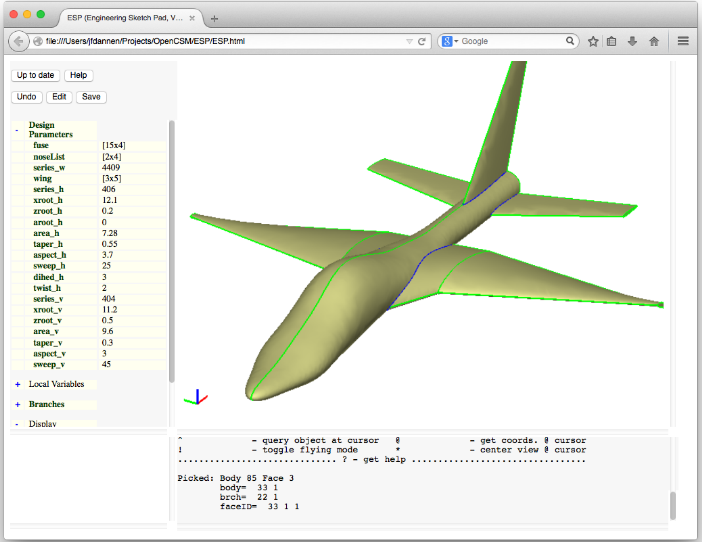
- A *Solid* Body is closed and manifold
- A *Sheet* Body is either open and/or non-manifold
- A *Wire* Body has no Faces

OpenCSM Script Example

```
# design parameters
despmtr  width        10.00
despmtr  depth         4.00
despmtr  height       15.00
despmtr  neckDiam      2.50
despmtr  neckHeight    3.00
despmtr  wall          0.20
despmtr  filRad1       0.25
despmtr  filRad2       0.10

# basic bottle shape (filleted)
set         baseHt   height-neckHeight
skbeg       -width/2 -depth/4 0
   cirarc   0        -depth/2 0  +width/2 -depth/4  0
   linseg   +width/2 +depth/4 0
   cirarc   0        +depth/2 0  -width/2 +depth/4  0
skend
extrude  0        0        baseHt
fillet   filRad1  0        0

# neck with a hole
set      holeBot height-neckHeight/2
cylinder 0   0   baseHt  0 0 height      neckDiam/2
cylinder 0   0   holeBot 0 0 height+wall neckDiam/2-wall
subtract

# join the neck to the bottle and apply a fillet at the union
union
fillet   filRad2  0        0
```



Screen Shot of ESP through serveCSM



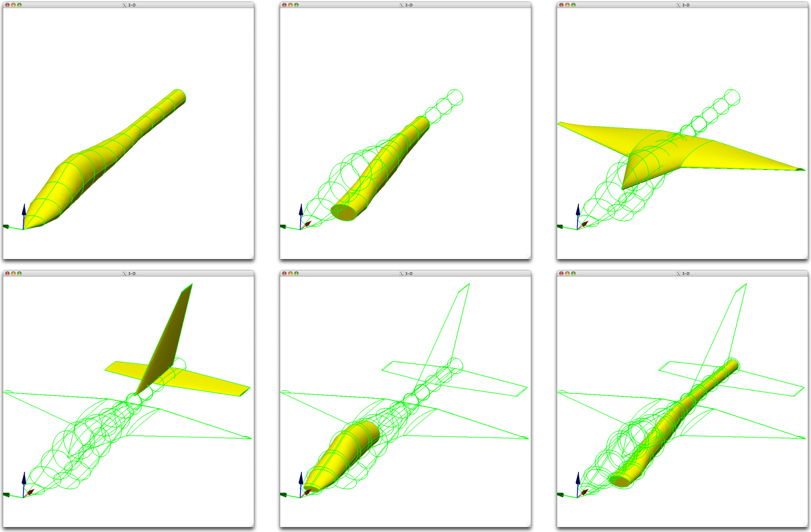
Solid Modeller

- Construction process guarantees that built models can be realizable Solids
  - watertight representation needed for 3D grid generators
  - Bodies of type WireBody and SheetBody are supported where needed
- Parametric models are defined in terms of:
  - Feature Tree
    - "recipe" for the construction of geometry
    - each "branch" specifies a *stack* operation
  - Design Parameters
    - "values" (dimension/sizing) that together describe a particular instance of the resultant build
    - can be scalar, vector or arrays
    - can have an associated "velocity"
  - *Internal* (driven) variables – in the form of mathematical expressions that depend on Design Parameters

Feature-based

- Configurations start with the generation of primitives
  - standard primitives: box, sphere, cone, cylinder, torus
  - grown primitives (from sketches): extrude, rule, blend, revolve, sweep, loft
  - user-defined primitives (UDPs)
- Body modification
  - transformations: translate, rotate, scale, mirror
  - applications: fillet, chamfer, hollow
  - user-defined functions (UDFs)
- Body combination
  - Booleans: intersect, subtract, union
  - other: join, connect, extract, combine

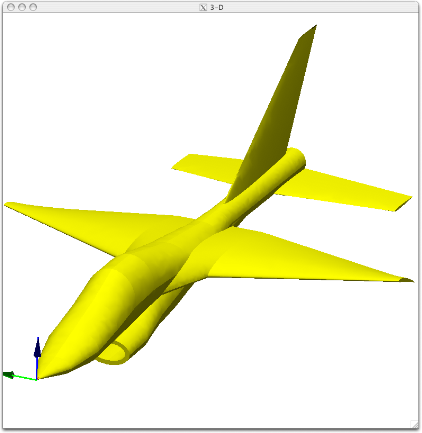
Build-up Sequence of Outer Mold Line



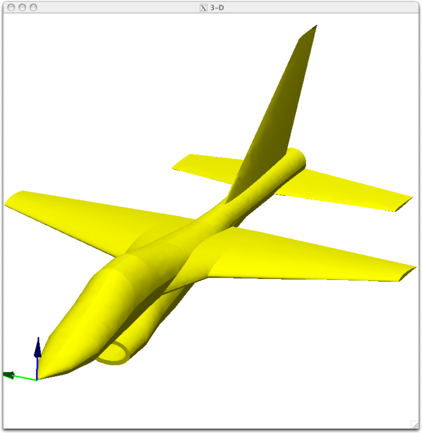
Parametric

- ESP models typically contain one or more Design Parameters
- Design Parameters can be single-valued, 1D vectors, or 2D arrays of numbers
- each Design Parameter has a current value, upper- and lower-bounds, and a current "velocity" (which is used to define sensitivities)
- Design Parameters can be "set" and "get"
  - through ESP's tree window
  - externally via calls to the API
- arguments of all operations can be written as "expressions" that can reference back to the Design Parameters

Parametric Variation 1: Untwisted Wing

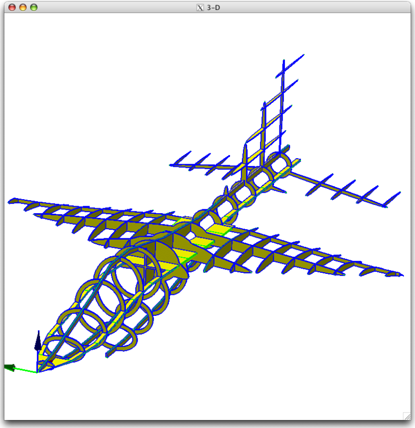


20° wing tip twist

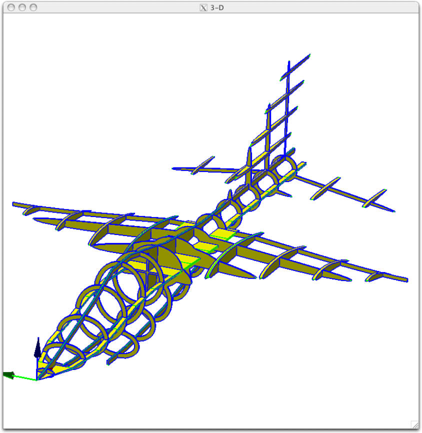


no wing tip twist

Parametric Variation 2: Fewer Ribs



8 thin wing ribs



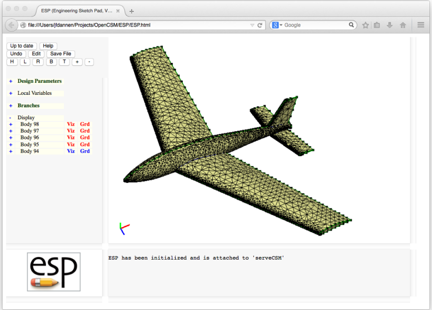
4 thick wing ribs

Distinguishing Feature: Associative

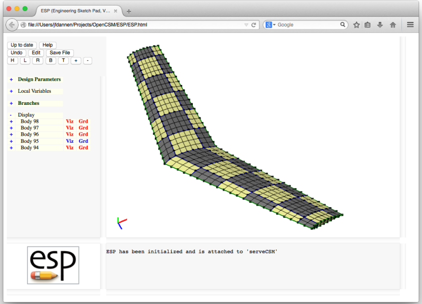
- ESP maintains a set of global and local attributes on a configuration that are persistent through rebuilds
    - the attributes are specified in the Feature Tree/CSM script
    - the attributes end up on generated Topology
- Supports the generation of multi-fidelity models
    - attributes can be used to associate conceptually-similar parts in the various models
- Supports the generation of multidisciplinary models
    - attributes can be used to associate surface groups which share common loads and displacements
- Supports the "marking" of Faces and Edges with ancillary info such as nominal grid spacings, material properties, ...

Multidisciplinary Models

- Driven by same Design Parameters
- Attributes provide "links" between models



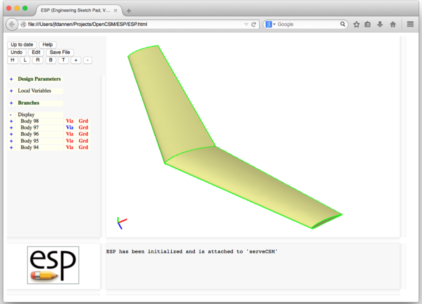
Outer mold line (OML)
for CFD



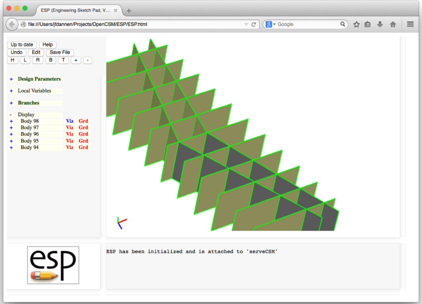
Built-up element (BEM)
for finite element method

Anatomy of Built-up Element Model

- Build two component models
- Intersect models to create trimmed structure
- Subtract waffle from OML to break into panels
- Union pieces for complete BEM

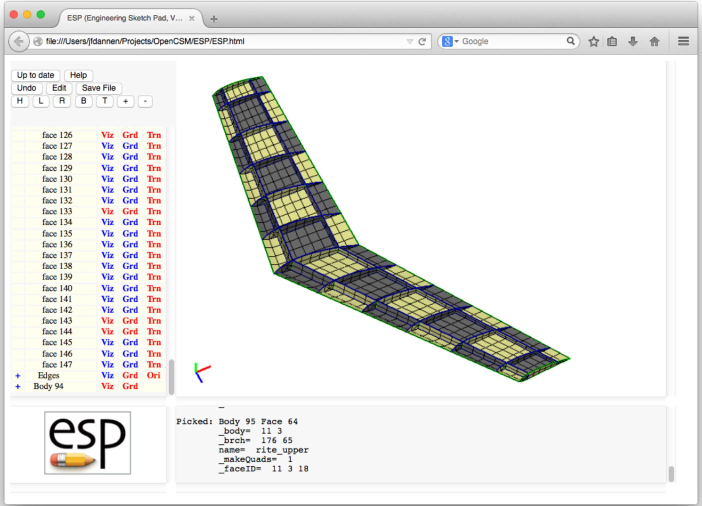


Outer mold line

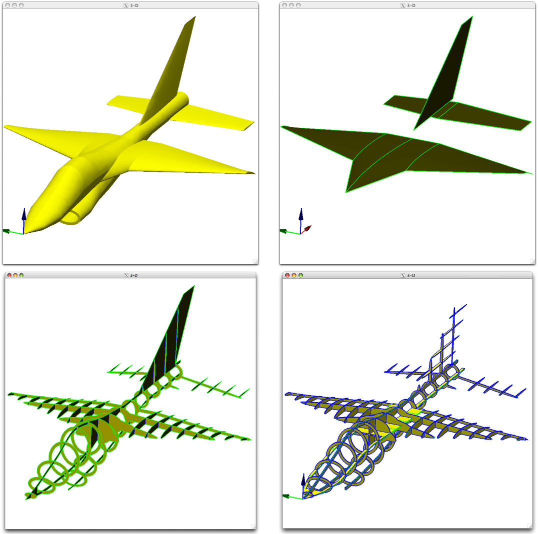


Waffle (untrimmed structure)

Interior view of Built-up Element Model



Multidisciplinary Models of a Fighter



Distinguishing Feature: Extensible

Users can add their own User-Defined Primitives

- UDP geometry construction can be written either *top-down*, *bottom-up* or both
- UDPs are EGADS applets
  - create and return EGADS Body or Node Objects
  - has access to the entire suite of methods provided by EGADS
  - written in C, C++, or FORTRAN, are compiled and built into Shared Objects/DLLs
- UDPs are coupled into ESP dynamically at run time

Users can add their own User-Defined Functions

UDFs are like UDPs except:

- can pull items off of the stack
- can return zero or more EGADS Body or Node Objects that will be pushed on the stack

Distinguishing Feature: Extensible

Users can add their own User-Defined Components

- UDCs can be thought of as “macros”
  and are found as separate files (from the *.csm* file)
- UDCs create zero or more stack entries
- UDCs are written as `CSM`-like scripts
  like routines, UDCs have *interface syntax* and specific *internal* variable scoping

OpenCSM UDPs, UDFs and UDCs

UDPs shipped with ESP

- NACA-4, -5, and -6 series airfoils
- Kulfan, Parsec, and Biconvex airfoils
- super-ellipse
- box (with rounded corners)
- Bezier surfaces and solids
- freeform surfaces and solids
- waffle
- import
- pod
- sew

UDCs shipped with ESP

- general rotation
- diamond airfoil
- flap
- spoiler
- popup
- fuselage
- wing
- duct
- strut

UDFs shipped with ESP

- bem
- poly
- attribute editing

Distinguishing Feature: Deployable

- ESP's back-end (server) runs on these compute platforms:
  - LINUX
  - Mac OSX
  - Windows 7 & above
- ESP's user-interface (client) runs in most modern web browsers:
  - FireFox
  - Google Chrome
  - Safari
  - Note: IE/Edge is not supported at this time
- ESP can be distributed just about anywhere
  - open-source project (using the LGPL 2.1 license) that is distributed as source
  - can be used in parallel compute environments
    EGADSlite being generated as part of a NASA NRA

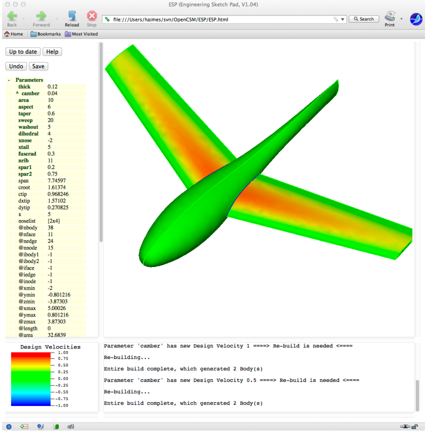
Distinguishing Feature: Embeddable

- Models are defined in `CSM` files/scripts
  - human readable ASCII
  - stack-like language is consistent with Feature Trees
  - contains looping and logical decisions
- `OpenCSM` modeling system is defined by an API that allows it to be embedded into other applications
- The `EGADS` API can be used once geometry is constructed to query attributes for BCs/material properties and perform meshing (evaluating the geometry directly)

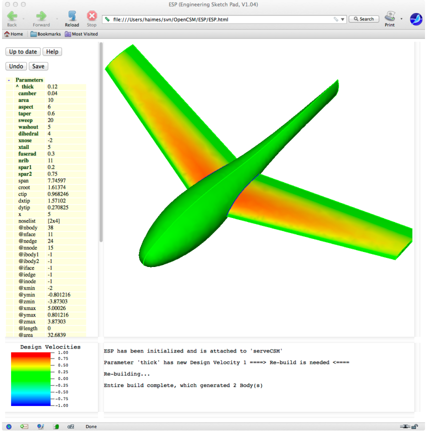
Distinguishing Feature: Differentiated

- `ESP` allows a user to compute the sensitivity of any part of a configuration with respect to any Design Parameter
  - *Configuration* and/or *Tessellation* sensitivities
- Much of `OpenCSM` has been analytically "differentiated"
  - efficient – since there is no need to regenerate the configuration
  - accurate – there is no truncation error as with "differencing"
- Compile-time code differentiation is used for some methods e.g. blend and some UDPs
- Other commands require the use of finite-differenced sensitivities
  - less efficient, since it requires the generation of a "perturbed" configuration
  - robust, an effective "mapping" technique guarantees the correct association of points in the baseline and perturbed geometries
  - less accurate, since one needs to carefully select a "perturbation step" that is a balance between truncation and round-off errors

Sensitivity Examples



Change in camber



Change in thickness

Computational Aircraft Prototype Synthesis

- Several MDO frameworks/environments have been developed over the last couple of decades
- These tend to focus on:
  - automating overall analysis process by creating “data flows” between user-supplied analysis packages
  - scheduling and dispatching of analysis execution
  - generation of suitable candidate designs via DOE, . . .
  - visualization of design spaces
  - improvements of designs via optimization
  - techniques for assessing and improving the robustness of designs

- "Data" that current MDO frameworks handle are "point" quantities
  - geometric parameters: length, thickness, camber, ...
  - operating conditions: speed, load, ...
  - performance values: cost, efficiency, range, ...
- No current framework handles "field" data directly
  - $xyz_{\text{verticalTail}}, \quad p_{\text{upperWing}}, \quad \Delta\vec{s}_{\text{fuselage}}$
  - example associated operations (consistent with the source):
    - copy (same as for "point" data)
    - interpolate/evaluate
    - integrate
    - supply the derivative
- Multidisciplinary coupling in current frameworks require that user supplies custom pairwise coupling routines

CAPS Background

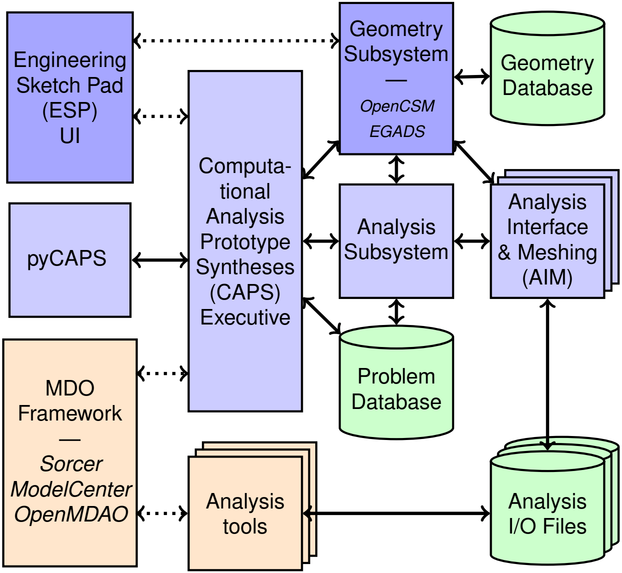
CAPS Goals

- Augment/fix MDO frameworks
- Provide the tools & techniques for generalizing analysis coupling
    - multidisciplinary coupling: aeroelastic, FSI
    - multi-fidelity coupling: conceptual and preliminary design
- Provide the tools & techniques for rigorously dealing with geometry (single and multi-fidelity) in a design framework / process

CAPS Access

- The main entry point into the CAPS system is the C/C++ API
- pyCAPS: Python interface for testing, demos and training

ESP with the CAPS Infrastructure



CAPS Executive

Responsible for:

- receiving commands from the framework/user, such as:
  - create a new Problem Database from an input model
  - set an operating condition
  - set a design parameter
  - make *linkages*
  - for each analysis tool:
    - create the inputs needed for analysis tool N
    - (analysis tool N is run by framework/user)
    - read the outputs from analysis tool N and store it in the Database
- dispatching commands to the Geometry and Analysis Subsystems
- initializing, reading from, and writing to the Problem Database
- communicating information back to the framework/user

Analysis Subsystem

Pre- and *Post*-Analysis — deals with the rich (“field”) data

Responsible for:

- getting BRep from the Geometry Subsystem
- performing grid generation for specified analysis – or – setting up for stand-alone meshing software
- calling the AIM plugin to set up for a specified analysis
- performing conservative transfers between different discrete representations of the same *boundaries*
- calling the AIM plugin to extract information from a specified analysis run

Note: Does NOT initiate analysis execution!

Analysis Interface & Meshing (AIM)

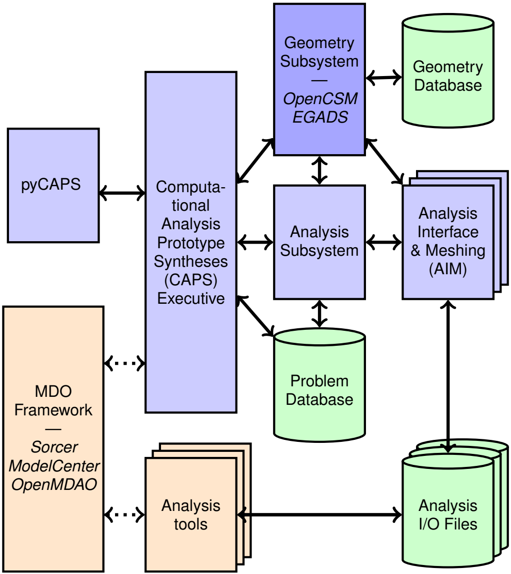
AIMs are `EGADS` applets (similar in concept to UDPs)

- Analysis identification – at AIM registration
  - number of inputs expected & number of possible outputs
  - geometric fidelities expected
- Analysis input generation – *Pre*
  - supplies Analysis Subsystem with information required to generate the input for the analysis (and optionally meshing)
    - format for the input file
    - possibly attribute BRep with geometric-based information
    - preparing the BRep data to be used for grid generation
  - plugin deals with populating the discrete BRep data from the mesh (the *bound* – "field" data)

Analysis Interface & Meshing (AIM)

- Analysis output parsing – *Post*
  - plugin deals with populating *bound*-based scalar, vector and/or state vector data from the solver run
  - reads or calculates integrated (performance) measures that can be used as objective functions for optimization (“point” data)
- Multidisciplinary coupling – when required
  - plugin provides functions to use the discrete data to Interpolate and/or Integrate (consistent with solver)
  - plugin provides *reverse* differentiated Interpolate and Integrate functions to facilitate conservative transfer optimization
  - automatically initiated in a *lazy* manner when the data transfer is requested

CAPS Infrastructure



Setup (or read) the Problem:

- Initialize Problem with *csm* (or *static*) file
  GeomIn and GeomOut parameters
- Specify *mission* parameters
- Make Analysis instances
  AnalysisIn and AnalysisOut params
- Create *Bounds*, *VetrexSets* & *DataSets*
- Establish linkages between parameters

Run the Problem:

- Adjust the appropriate parameters
- Regenerate Geometry (if *dirty*)
- Call for Analysis Input file generation
- Framework/user runs each *solver*
- Inform CAPS that an Analysis has run
  fills AnalysisOut params & *DataSets* (lazy)
- Generate *Objective Function*

Save the Problem DB (*checkpointing*)

Current Collection of AIMs

Low Fidelity

- AWAVE
- Friction
- AVL
- XFoil
- ASWING*

Structural Analysis

- mySTRAN
- NASTRAN
- ASTROS
- *Status:*
  - linear static & modal analysis
  - support for composites, optimization & aeroelasticity

3D CFD

- Meshing:
  - Surface
    - Native `EGADS`
    - AFLR4
    - Pointwise*
  - Volume
    - TetGen
    - AFLR3
    - Pointwise*
- Solvers:
  - Cart3D
  - Fun3D
  - SU^2
  - **SANS***

Closing Remarks – Future Directions

PAGODA

Develop a distributed/threaded geometry system to support solver meshing, adaptation, and sensitivities for analysis and design

DARPA's TRADES Program – Jan Vandenbrande, PM

The TRAnsformative DESign (TRADES) program aims to advance the foundational mathematics and computational tools required to generate and better manage the enormous complexity of design.

- *Design Responding to Engineering Analysis in support of Manufacturing – DREAM*
  - Fully couple conceptual optimization to the following phases
  - Embrace volumetric representations (VReps) in design
- *Augmented Design Through Analysis and Visualization – Facilitating Better Designs and Enhanced Designers*

Closing Remarks – On The Origins of CAD

TECHNICAL MEMORANDUM Copy No. 14

COMPUTER-AIDED DESIGN RELATED TO
THE ENGINEERING DESIGN PROCESS

by

S. A. Coons and R. W. Mann
(Mechanical Engineering Department)

8436-TM-5

October, 1960

Contract No. AF-33(600)-40604

The work reported in this document has been made possible through the support extended to the Massachusetts Institute of Technology, Electronic Systems Laboratory, by the Manufacturing Methods Division, AMC Aeronautical Systems Center, United States Air Force, under Contract No. AF-33(600)-40604, M. I. T. Project No. 8436. Part of the work is being performed by the Mechanical Engineering Department, Design and Graphics Division, under M. I. T. Project No. 8477. The report is published for technical information only and does not represent recommendations or conclusions of the sponsoring agency.

Approved by: Douglas T. Ross, Project Engineer
Head, Computer Applications Group

Electronic Systems Laboratory
Department of Electrical Engineering
Massachusetts Institute of Technology
Cambridge 39, Massachusetts

Engineering Design Process

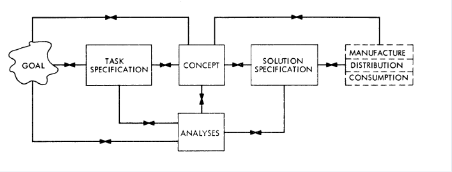


Figure #1

Acknowledgements

Supported by AFRL Contract FA8050-14-C-2472

Computational Aircraft Prototype Synthesis (CAPS)

- Ed Alyanak (AFRL), Technical Monitor
- Team: Ryan Durscher, Nitin Bhagat, Darcy Allison

Supported by NASA Aeronautics NRA #NNX16AQ15A

PAGODA: PArallel GeOmetry for Design and Analysis

- Bill Jones (LaRC), Technical Monitor

Software available at:

http://acdl.mit.edu/ESP